PSYS.COM
โปรแกรมช่วยก๊อบปี้ซิสเต็มสำหรับแผ่นข้อมูล

ขจร โรจนเมธินทร์

ทุกครั้งที่คุณนำดิสค์แผ่นใหม่มาใช้งานคุณจะต้องตัดสินใจว่าจะฟอร์แมตดิสค์แผ่นนั้นให้มีซิสเต็มไฟล์หรือไม่ ซึ่งถ้าคุณเลือกแบบมีซิสเต็มไฟล์ คุณก็จะเสียเนื้อที่บนดิสค์สำหรับเก็บไฟล์ IBMBIO.COM, IBMDOS.COM, COMMAND.COM ไปไม่น้อยกว่า 68 กิโลไบต์ อย่างไรก็ตามการมีซิสเต็มไฟล์บนดิสค์ทำให้มีความสะดวกในการใช้งานคือ คุณสามารถใช้ดิสค์แผ่นนั้นบูตเครื่องได้ และช่วยให้ไม่ต้องเปลี่ยนดิสค์ใหม่เมื่อเครื่องต้องการอ่านส่วน Transient ของ COMMAND.COM ลงบนหน่วยความจำ

คราวนี้หากคุณเลือกฟอร์แมตแบบไม่มีซิสเต็มไฟล์ แล้วคุณเกิดเปลี่ยนใจอยากให้ดิสค์แผ่นนั้นมีซิสเต็มไฟล์ คุณอาจใช้โปรแกรม SYS.COM ของดอสช่วยเพิ่มซิสเต็มไฟล์ลงบนดิสค์ได้ แต่ SYS.COM จะเพิ่มซิสเต็มไฟล์ได้ในกรณีที่ดิสค์แผ่นนั้นยังไม่มีข้อมูลเก็บอยู่เท่านั้น โปรแกรม PSYS.COM จะช่วยย้ายข้อมูลบนดิสค์ใหม่โดยทำให้เกิดที่ว่างพอสำหรับซิสเต็มไฟล์ จากนั้นคุณก็สามารถใช้ SYS.COM เพิ่มซิสเต็มไฟล์ได้

```
PSYS B:
IF ERRORLEVEL 1 GOTO END
SYS B:
COPY COMMAND.COM B:
:END
```

รูปที่ 1 แสดงคำสั่งในแบตช์ไฟล์

แนะนำตัว PSYS.COM

PSYS.COM จะจัดเนื้อที่บนดิสค์ให้เกิดที่ว่างสำหรับซิสเต็มไฟล์โดยจะตรวจสอบที่ว่างบนดิสค์ก่อนว่ามีเหลือพอสำหรับซิสเต็มไฟล์หรือไม่ หากไม่พอก็จะยกเลิกการทำงานและถ้าดิสค์แผ่นนั้นมีซิสเต็มไฟล์อยู่แล้วคุณก็สามารถลบซิสเต็มไฟล์เก่าได้ ในขณะที่ PSYS.COM ทำงานก็จะตรวจข้อผิดพลาดต่าง ๆ ที่อาจมีบนดิสค์ของคุณ เช่น มีเซกเตอร์เสีย, มีข้อผิดพลาดใน File Allocation Table (FAT) เป็นต้น หากพบข้อผิดพลาด PSYS.COM ก็จะรายงานให้ทราบทันที

หลังจากที่ PSYS.COM ทำงานเรียบร้อยแล้ว คุณจะต้องใช้ SYS.COM เพิ่มซิสเต็มไฟล์ต่อทันที หากคุณพยายามที่จะใช้ดิสค์แผ่นนั้นก่อนจะเพิ่มซิสเต็มไฟล์ดอสจะคิดว่าคุณกำลังใช้ดิสค์แผ่นใหม่ซึ่งยังไม่มีข้อมูล ทั้งนี้เพราะ PSYS.COM ทำการแก้ไข FAT และดิสค์ไดเรคทอรีใหม่เพื่อเตรียมให้ SYS.COM ทำงานต่อและเมื่อ SYS.COM ทำงานเสร็จ ขั้นต่อไปคุณก็ต้องก๊อบปี้ COMMAND.COM มาไว้บนดิสค์แผ่นนั้นด้วย คุณก็จะได้ดิสค์ที่มีซิสเต็มมาใช้งานทันที

เพื่อความสะดวกคุณอาจสร้างแบตช์ไฟล์ขึ้นมาให้ทำหน้าที่ตามขั้นตอนเหล่านี้ดังรูปที่ 1 ทั้งนี้เพราะ PSYS.COM จะคืนค่าให้พารามิเตอร์ ERRORLEVEL ซึ่งถ้ามีข้อผิดพลาดเกิด พารามิเตอร์ ERRORLEVEL จะมีค่าเป็น 1

โปรแกรม PSYS.COM ถูกเขียนให้ทำงานได้กับฟลอปปี้ดิสค์ขนาด 5.25

นิ้ว แบบ 40 แทร็ก แต่ละแทร็กแบ่งออกเป็น 8 หรือ 9 เซกเตอร์ต่อแทร็ก เก็บได้ 1 หรือ 2 หน้าเท่านั้น หากคุณใช้ PSYS.COM กับดิสค์แบบอื่น โปรแกรมจะตรวจพบข้อผิดพลาด และยกเลิกการทำงานทันที (ตัวโปรแกรมดังรูปที่ 2)

ข้อจำกัดอื่น ๆ สำหรับ PSYS.COM ก็คือ ไม่ควรใช้ PSYS.COM กับดิสค์ที่มีสับไดเรคทอรี และไม่ควรใช้ PSYS.COM กับดิสค์ที่ป้องกันการก๊อบปี้ เนื่องจากดิสค์เหล่านี้มักจะมีฟอร์แมตที่แตกต่างไปจากฟอร์แมตของดอส

มารู้จักกับ File Management ของดอส

หน้าที่สำคัญอันหนึ่งของระบบปฏิบัติการก็คือ การจัดระเบียบของข้อมูลบนดิสค์ โดยจัดเนื้อที่ให้แต่ละไฟล์ไม่ซ้ำกัน ซึ่งผมจะขอกล่าวถึงเฉพาะฟลอปปี้ดิสค์เท่านั้น

ดอสแบ่งชนิดของฟลอปปี้ดิสค์ออกเป็น 4 ชนิดดังนี้คือ

1. เก็บได้ 1 หน้า และแต่ละแทร็กแบ่งออกเป็น 8 เซกเตอร์
2. เก็บได้ 2 หน้า และแต่ละแทร็กแบ่งออกเป็น 8 เซกเตอร์
3. เก็บได้ 1 หน้า และแต่ละแทร็กแบ่งออกเป็น 9 เซกเตอร์
4. เก็บได้ 2 หน้า และแต่ละแทร็กแบ่งออกเป็น 9 เซกเตอร์

ดิสค์แต่ละแผ่นจะมี 40 แทร็ก และเมื่อเราฟอร์แมตดิสค์แต่ละแทร็กก็จะแบ่งออกเป็นเซกเตอร์ขนาด 512 ไบต์ ดังนั้นดิสค์แบบที่ 4 ก็จะมี 720 เซกเตอร์ และมีความจุ 360 กิโลไบต์ โดยดอสจะแบ่งพื้นที่บนดิสค์เพื่อใช้เก็บบูตเซกเตอร์ 1 เซกเตอร์, ดิสค์ไดเรคทอรี 7 เซกเตอร์, FAT 4 เซกเตอร์ (โดยเก็บ

รูปที่ 2
โปรแกรม PSYS.ASM

```
code            segment para public 'code'
                assume cs:code,ds:code,es:code
                org 100h
begin:          jmp prep
;
copy_right      db 13,10,'Copyright 1989 KJ''s software',13,10,'$'
errmsg1         db 13,10,'Invalid Drive Specifier',13,10,'$'
errmsg2         db 13,10,'Illegal Disk Format',13,10,'$'
errmsg3         db 13,10,'Disk Read Error',13,10,'$'
errmsg4         db 13,10,'Disk Write Error',13,10,'$'
errmsg5         db 13,10,'Insufficient Space in Disk Directory',13,10,'$'
errmsg6         db 13,10,'Insufficient Disk Space',13,10,'$'
errmsg7         db 13,10,'Allocation Chain Error',13,10,'$'
errmsg8         db 13,10,'Cannot Delete System Files',13,10,'$'
errmsg9         db 13,10,'Bad Sector in System Area',13,10,'$'
msg1      db 13,10,'System Perp Completed',13,10,'$'
msg2      db 13,10,'System Already Installed - Delete Old System (Y/N)?',13,10,'$'
msg3      db 13,10,'Execution Aborted',13,10,'$'
;
last_cluster        dw 41               ;last available cluster found
sec_per_cluster     dw ?                ;number of sector per cluster
clusters            dw ?                ;total number of data clusters on disk
DIR_entries         dw ?                ;maximum number of directory entries
DIR_sectors         dw ?                ;number of sectors used for directory
FAT_sectors         dw ?                ;number of sectors used for one copy of the FAT
FAT_entry           dw ?                ;temporary storage for any FAT entry
drive               db ?                ;current disk drive
format_table        dw 64,4,2           ;S-9 disk format parameter
                    dw 112,7,2          ;D-9   "       "
                    dw 64,4,1           ;S-8   "       "
                    dw 112,7,1          ;D-8   "       "
system_id       db 'IBMBIO',32,32,'COM' ;text of system file IBMBIO.COM
ibmbio          db 'B:IBMBIO.COM',0     ;ASCIIZ string for DOS file functions
ibmdos          db 'B:IBMDOS.COM',0     ;  "       "       "
cbuffer         dw offset endprog       ;pointer to buffer area for cluster transfer
FAT_data        dw offset endprog+1024  ;pointer to area where FAT image is stored
DIR_data        dw offset endprog+2048  ;pointer to area where DIR image is stored
;
prep            proc near               ;start of main program
;
;Parse the command line for a drive specifier. If none, use default drive.
;
                cld                     ;clear DF for string operation
                mov si,80h              ;point SI to start of command line
                lodsb                   ;get number of characters entered
```

```
                cmp al,1                ;one or less characters specified?
                jbe prep1               ;yes, then use default drive
                inc si                  ;skip first delimeter
                lodsb                   ;get drive specifier
                and al,0DFh             ;capitalize it
                sub al,'A'              ;convert ASCII to drive number
                cmp al,1                ;is drive specified A: or B:?
                jbe prep2               ;yes, then finish
                lea dx,errmsg1          ;no, then error
                jmp error_exit          ;display message and abort
prep1:          mov ah,19h              ;get current drive function
                int 21h
                cmp al,1                ;is it drive A: or B:?
                jbe prep2               ;yes, skip error routine
                lea dx,errmsg1

;
;Now program execution is routed here when a critical error is detected.
;
error_exit:     mov ah,9                ;print string function
                int 21h
                mov ah,2                ;print a character
                mov dl,7                ;beep
                int 21h
error1:         mov ah,4Ch              ;terminate program function
                mov al,1                ;set ERRORLEVEL return code to 1
                int 21h
;
;Check the format of the disk to verify that it's a 40-track diskette
;(8 or 9 sectors per track, one or two sides).
;Store information pertaining to the disk format.
;
prep2:          mov drive,al            ;save current drive ID
                mov dl,al               ;get drive in DL
                inc dl                  ;adjust drive ID for DOS function call
                push ds                 ;save segmetn register
                mov ah,1Ch              ;get FAT information
                int 21h
                mov cl,[bx]             ;store ID byte in CL
                pop ds                  ;restore DS
                cmp cl,0FCh             ;ID byte greater than or equal to FCh?
                jae prep2a              ;yes, then format is ok
                lea dx,errmsg2          ;no, then format is invalid
                jmp error_exit          ;abort on error
prep2a:         cbw                     ;convert byte in AL to word in AX
                mov sec_per_cluster,ax  ;store number of sectors per cluster
                mov clusters,dx         ;store total number of data clusters
                sub cl,0FCh             ;normalize ID byte by subtracting FCh
                mov al,cl
                mov dl,6
                mul dl                  ;multiply by 6
                mov bl,al               ;transfer table index in AL to BL
                xor bh,bh               ;convert byte to word
```

→

```
                add bx,offset format_table   ;form address of disk format parameter line
                mov ax,[bx]             ;read number of dir entries from table
                mov DIR_entries,ax      ;save number of entries
                mov ax,[bx+2]           ;read directory length from table
                mov DIR_sectors,ax      ;save directory length
                mov ax,[bx+4]           ;read FAT length from table
                mov FAT_sectors,ax      ;save FAT length
;
;Read root directory and one copy of FAT into memory from the selected drive.
;
prep3:          call read_FAT           ;read in one copy of the file allocation table
                jnc prep3a              ;continue if no error on read
                lea dx,errmsg3          ;exit if error occured from reading
                jmp error_exit
prep3a:         call read_DIR           ;read in disk directory
                jnc prep4               ;continue if no error
                lea dx,errmsg3
                jmp error_exit
;
;Determine if the disk is already system formatted by seeing if the first
;file in the directory is IBMBIO.COM.
;
prep4:          lea si,system_id        ;point SI to 'IBMBIO.COM'

                mov di,DIR_data         ;point DI to first filename
                mov cx,11               ;11 bytes to check
                repe cmpsb              ;compare the two string
                jne prep6               ;skip delete routine if no system
;
;Disk already system formatted--see if user wants to delete the system file.
;
                lea dx,msg2             ;get query message address
                mov ah,9                ;print string function
                int 21h
prep5:          mov ah,0                ;get key press
                int 16h
                and al,0DFh             ;capitalize it
                cmp al,'Y'              ;was 'Y' pressed?
                je prep5a               ;yes, delete system file
                cmp al,'N'              ;was 'N' pressed?
                jne prep5               ;no, then get another keypress
                lea dx,msg3             ;get abort message addres
                mov ah,9                ;print message
                int 21h
                jmp error1              ;exit with errorlevel 1
;
;Delete system files - change file attributes to 0 (eliminating read-only bit)
;before using DOS delete function.
;
prep5a:         cmp drive,1             ;is B: the current drive?
                je prep5b               ;yes, continue
                dec ibmbio              ;no, set ASCIIZ drive designator to A:
                dec ibmdos
```

→

```
prep5b:     lea dx,ibmbio           ;change IBMBIO.COM file attribute for deletion
            mov ah,43h              ;change file attribute
            mov cx,0                ;new file attribute
            mov al,1                ;specify that attribute is to be changed
            int 21h
            jc prep5c               ;exit on error if unsuccessful
            lea dx,ibmdos           ;do the same
            mov ah,43h
            mov cx,0
            mov al,1
            int 21h
            jc prep5c
            lea dx,ibmbio           ;now delete IBMBIO.COM
            mov ah,41h              ;delete file function
            int 21h
            jc prep5c
            lea dx,ibmdos
            mov ah,41h
            int 21h
            jc prep5c
            jmp prep3               ;reread FAT and directory
prep5c:     lea dx,errmsg8          ;files cannot be deleted
            jmp error_exit
;
;Check that the disk has adequate free space before continuing.
;
prepc:      mov dl,drive            ;get drive in DL
            inc dl                  ;adjust drive ID for DOS function call
            mov ah,36h              ;get free space function
            int 21h
            mov ax,sec_per_cluster  ;get sector per cluster

            mul bx                  ;multuply to get free sectors
            cmp ax,100              ;at least 100 sectors free?
            jae prep7               ;yes, then continue
            lea dx,errmsg6          ;no, then load error message
            jmp error_exit
;
;Check first two directory entries to see whether each is used or available.
;For each one not available, transfer it to the first available space.
;
prep7:      mov si,DIR_data         ;point SI to directory image
            cmp byte ptr [si],0     ;is the first entry available?
            je prep9                ;yes, then goto next entry
            cmp byte ptr [si],0E5h  ;erased file?
            jne prep7a              ;no, then move it
            mov byte ptr [si],0     ;mark entry as 'Never used'
            jmp prep9               ;jump to next entry
prep7a:     mov dx,2                ;entry is occupied
            call find_entry         ;find first available directory entry
            jnc prep8               ;entry found - continue
            lea dx,errmsg5          ;no space in dir
            jmp error_exit
```

→

```
prep8:      mov ax,0
            call move_entry         ;copy entry 0 into first availability slot
prep9:      mov si,DIR_data         ;check second entry for availability
            add si,32               ;point SI to second entry
            cmp byte ptr [si],0     ;first byte zero?
            je prep13               ;yes, then it's available
            cmp byte ptr [si],0E5h  ;erased file?
            jne prep9a              ;no, then move it
            mov byte ptr [si],0     ;mark entry
            jmp prep13              ;continue
prep9a:     mov dx,2                ;not available
            call find_entry
            jnc prep10              ;continue if empty slot found
            lea dx,errmsg5
            jmp error_exit
prep10:     mov ax,1
            call move_entry
;
;Copy each of the first 40 clusters currently in use to the clusters beyond the
;first 40  and adjust the corresponding FAT entries accordingly.
;
prep13:     mov cx,40               ;check 40 clusters
            mov dx,2                ;start checking at cluter 2
prep14:     push cx                 ;save loop counter
            call get_FAT_entry      ;get FAT entry current cluster
            cmp ax,0                ;is it available?
            je prep17               ;yes, then proceed to next cluster
            cmp ax,0FF0h            ;is FAT entry less than FF0h?
            jb prep14a              ;yes, then move cluster
            cmp ax,0FF7h            ;is it greater than FF7h?
            ja prep14a              ;yes, then move cluster
            pop cx                  ;bad cluster - clean up stack
            lea dx,errmsg9
            jmp error_exit
prep14a:    mov FAT_entry,ax        ;save FAT entry for later
            mov ax,last_cluster     ;get number of last available cluster
            inc ax                  ;next one to egin search
            mov bx,0
            call find_cluster       ;find first cluster with FAT entry of zero
            mov last_cluster,ax     ;store cluster number just found
            call move_cluster       ;copy the old cluster to the new
            jnc prep15              ;continue if no error in disk I/O
            pop cx                  ;on error, clean up stack
            jmp error_exit
prep15:     mov ax,FAT_entry        ;retrieve the FAT entry for the old cluster
            push dx                 ;save the old cluster number
            mov dx,last_cluster     ;target the new cluster
            call put_FAT_entry      ;put the old FAT entry into the new
            pop dx                  ;restore the old cluster number
            mov ax,0                ;mark the old cluster available
            call put_FAT_entry
            mov bx,dx               ;get old cluster number to BX for search
```

→

```
254             mov ax,2                ;start search at cluster 2
255             call find_cluster       ;find FAT entry that referenced old cluster
256             jnc prep16              ;if found then continue
257             call search_DIR         ;if not, then look in the directory
258             jnc prep15a             ;if found, then continue
259             pop cx
260             lea dx,errmsg7
261             jmp error_exit
262 prep15a:    mov ax,last_cluster     ;get new cluster number
263             mov [bx],ax
264             jmp prep17
265 prep16:     push dx
266             mov dx,ax               ;change FAT entry that formerly reference
267             mov ax,last_cluster
268             call put_FAT_entry
269             pop dx
270 prep17:     pop cx
271             inc dx                  ;index to sequential cluster
272             loop prep14             ;loop until the first 40 cluster are clear
273 ;
274 ;Write two copies of the file allocation table and one copy of the disk
275 ;directory back to disk, then exit
276 ;
277 prep17a:    call write_FAT          ;write two copies of FAT
278             jnc prep18              ;continue if no error on write
279             lea dx,errmsg4
280             jmp error_exit
281 prep18:     call write_DIR          ;write the directory to disk
282             jnc prep19
283             lea dx,errmsg4
284             jmp error_exit
285 prep19:     lea dx,msg1
286             mov ah,9
287             int 21h
288             mov ax,4C00h            ;terminate program with return code 0
289             int 21h
290 prep        endp
291 ;
292 ;-----------------------------------------------------------------------------
293 ; GET_FAT_ENTRY routine returns the FAT entry for the indicated cluster.
294 ; Entry : DX - cluster number (2-XXX)
295 ; Exit  : AX - FAT entry
296 ;-----------------------------------------------------------------------------
297 get_FAT_entry proc near
298             mov ax,3                ;multiply cluster number by 3
299             push dx                 ;save cluster number
300             mul dx                  ;multiply
301             pop dx                  ;restore cluster number
302             shr ax,1                ;divide by 2 to obtain offset into FAT
303             mov bx,ax               ;transfer to BX
304             add bx,FAT_data         ;make address relative to code segment
305             mov ax,[bx]             ;get word
306             test dx,1               ;is cluster number odd or even?
307             je getf1                ;even, then jump
308             mov cl,4                ;odd, then keep upper 12 bits of word
309             shr ax,cl               ;by shifting right 4 bits
310             ret
311 getf1:      and ax,0FFFh            ;even number, then mask off upper 4 bits
312             ret
313 get_FAT_entry endp
314 ;
315 ;-----------------------------------------------------------------------------
316 ; PUT_FAT_ENTRY places the designated number into the FAT entry for the named
317 ; cluster.
318 ; Entry : DX - cluster number (2-XXX)
319 ;         AX - entry
320 ;-----------------------------------------------------------------------------
321 put_FAT_entry proc near
322             push ax                 ;save new FAT entry
323             push dx                 ;save cluster number
324             mov ax,3                ;multiply cluster by 3
325             mul dx
326             pop dx                  ;restore cluster number
327             shr ax,1                ;divide by 2
328             mov bx,ax               ;transfer to BX
329             add bx,FAT_data         ;complete offset address of FAT entry
330             pop ax                  ;restore FAT entry
331             test dx,1               ;is cluster odd or even?
332             je putf1                ;even, then jump
333             mov cl,4                ;odd - shift entry 4 bits left
334             shl ax,cl
335             and word ptr [bx],0Fh   ;zero upper 12 bits of word
336             or [bx],ax              ;install new FAT entry
337             ret
338 putf1:      and [bx],0F000h         ;even - zero lower 12 bits
339             or [bx],ax              ;place entry in lower 12 bits
340             ret
341 put_FAT_entry endp
342 ;
343 ;-----------------------------------------------------------------------------
344 ; The following four  routines write and read complete copies of the FAT and
345 ; the disk directory to and from the selected drive.
346 ;-----------------------------------------------------------------------------
347 read_FAT    proc near               ;read one copy of the FAT into memory
348             mov al,drive            ;define drive
349             mov dx,1                ;start at sector 1
350             mov cx,FAT_sectors      ;set number of sectors
351             mov bx,FAT_data         ;define buffer address
352             int 25h                 ;DOS absolute sector read
353             pop dx                  ;clean flag of stack
354             ret
355 read_FAT    endp
356 ;
357 read_DIR    proc near               ;read one copy of directory into memory
```

```
358             mov al,drive            ;define drive
359             mov dx,FAT_sectors      ;get number of sectors per FAT
360             shl dx,1                ;double it for two copies of the FAT
361             inc dx                  ;directory starts in next sector
362             mov cx,DIR_sectors      ;number of sectors to read
363             mov bx,DIR_data         ;address of storage buffer
364             int 25h                 ;DOS absolute sector read
365             pop dx                  ;clean flag of stack
366             ret
367 read_DIR    endp
368 ;
369 write_FAT   proc near              ;write two copies of the FAT to disk
370             mov al,drive            ;define drive
371             mov dx,1                ;begin at sector 1
372             mov cx,FAT_sectors      ;get number of sectors per FAT
373             mov bx,FAT_data         ;define buffer address
374             int 26h                 ;DOS absolute sector write
375             pop dx                  ;clean flag of stack
376             jc wfat1                ;exit on write error
377             mov al,drive            ;repeat for second copy of FAT
378             mov dx,1
379             mov cx,FAT_sectors
380             add dx,cx
381             mov bx,FAT_data
382             int 26h
383             pop dx
384 wfat1:      ret
385 write_FAT   endp
386 ;
387 write_DIR   proc near              ;write disk directory image in memory to disk
388             mov al,drive            ;define drive
389             mov dx,FAT_sectors      ;get number of sectors per FAT
390             shl dx,1                ;double it for two copies
391             inc dx                  ;directory starts in next sector
392             mov cx,DIR_sectors      ;number of sectors to read
393             mov bx,DIR_data         ;address of data to be written
394             int 26h                 ;DOS absolute sector write
395             pop dx                  ;clean flag of stack
396             ret
397 write_DIR   endp
398 ;
399 ;-------------------------------------------------------------------------
400 ; FIND_ENTRY searches thru the image of the disk directory stored in memory
401 ; and finds first available directory entry.
402 ; Entry : DX - starting entry (0-XXX)
403 ; Exit  : DX - first available entry
404 ;         CF - clear : available entry found
405 ;              set   : not found
406 ;-------------------------------------------------------------------------
407 find_entry  proc near
408             mov cx,DIR_entries      ;set maximum number of tries in CX
409             sub cx,dx
```

→

```
410             push cx                 ;save count value
411             mov bx,dx               ;get starting entry number in BX
412             mov cl,5                ;multiply BX by 32
413             shl bx,cl
414             add bx,DIR_data         ;complete offset address
415             pop cx                  ;restore loop count
416 findent1:  cmp byte ptr [bx],0     ;is the first byte zero?
417             je findent2             ;yes, then it's available
418             cmp byte ptr [bx],0E5h  ;is it erased file?
419             je findent2             ;yes, then it's available
420             inc dx                  ;not available - check next entry
421             add bx,32               ;address of next entry
422             loop findent1           ;loop back if some remain
423             stc                     ;none available - set CF
424             ret
425 findent2:  clc                     ;entry found - clear CF
426             ret
427 find_entry  endp
428 ;
429 ;-------------------------------------------------------------------------
430 ; MOVE_ENTRY routine transfers a directory entry from one space to another
431 ; and sets the old entry to indicate that it's available.
432 ; Entry : AX - source entry (0-XXX)
433 ;         DX - target entry (0-XXX)
434 ;-------------------------------------------------------------------------
435 move_entry  proc near
436             mov si,ax               ;convert entry number to offset address in SI
437             mov cl,5
438             shl si,cl
439             add si,DIR_data
440             mov di,dx
441             mov cl,5
442             shl di,cl
443             add di,DIR_data
444             push si                 ;save address of old entry for late
445             mov cx,32               ;32 bytes per entry
446             rep movsb               ;move all 32 bytes
447             pop si                  ;get address of old entry
448             mov byte ptr [si],0     ;mark entry as 'Never used'
449             ret
450 move_entry  endp
451 ;
452 ;-------------------------------------------------------------------------
453 ; FIND_CLUSTER routine finds the first cluster whose FAT entry matches the
454 ; number specified in BX.
455 ; Entry : AX - starting cluster (2-XXX)
456 ;         BX - FAT entry
457 ; Exit  : AX - first cluster found
458 ;         CF - clear : cluster found
459 ;              set   : not found
460 ;-------------------------------------------------------------------------
461 find_cluster  proc near
```

→

```
462             mov cx,clusters         ;get number of clusters
463             add cx,2                ;calculate max number of search loops
464             sub cx,ax
465             push dx                 ;save DX
466             mov dx,ax               ;starting cluster in DX
467 findcls1:  push cx                 ;save count of cluster s checked
468             push bx                 ;save FAT entry value
469             call get_FAT_entry      ;get FAT entry for designated cluster
470             pop bx                  ;restore FAT entry to BX
471             pop cx                  ;restore count
472             cmp ax,bx               ;is it what we're searching for?
473             je findcls2             ;yes, then terminate search
474             inc dx                  ;no, then try next cluster
475             loop findcls1
476             pop dx                  ;restore DX
477             stc                     ;set CF to indicate cluster not found
478             ret
479 findcls2:  mov ax,dx               ;put cluster number in AX
480             pop dx                  ;restore DX
481             clc                     ;clear CF to indicate success
482             ret
483 find_cluster  endp
484 ;
485 ;---------------------------------------------------------------------------
486 ; CLS2SEC routine returns the sector number that corresponds to the cluster
487 ; number input.
488 ; Entry : DX - cluster number (2-XXX)
489 ; Exit  : DX - sector number
490 ;---------------------------------------------------------------------------
491 cls2sec     proc near
492             sub dx,2                ;subtract two from the cluster number
493             mov ax,sec_per_cluster  ;get number of sectors in each cluster
494             mul dx                  ;multiply DX by sectors in each cluster
495             mov dx,ax               ;transfer to DX
496             mov ax,FAT_sectors      ;get number of sectors in FAT
497             shl ax,1                ;double it for two copies of the FAT
498             add dx,ax               ;add FART sector
499             add dx,DIR_sectors      ;add directory sectors
500             inc dx                  ;data starts in next sector
501             ret
502 cls2sec     endp
503 ;
504 ;---------------------------------------------------------------------------
505 ; MOVE_CLUSTER routine copies the contents of the source cluster into the
506 ; target cluster.
507 ; Entry : DX - source cluster (2-XXX)
508 ;         AX - target cluster (2-XXX)
509 ; Exit  : CF - clear : no error in disk I/O
510 ;              set   : I/O error
511 ;---------------------------------------------------------------------------
512 move_cluster  proc near
513             push dx
```

→

```
514             push ax
515             call cls2sec            ;get sector number of source cluster
516             mov cx,sec_per_cluster  ;number of sector to read
517             mov al,drive            ;specify drive
518             mov bx,cbuffer          ;set data transfer area
519             int 25h                 ;DOS sector read
520             pop ax
521             pop dx
522             jc movec1               ;exit on error
523             call cls2sec            ;get sector number of target cluster
524             mov cx,sec_per_cluster  ;number of sector to read
525             mov al,drive            ;specify drive
526             mov bx,cbuffer          ;set data transfer area
527             int 26h                 ;DOS sector write
528             pop ax
529             pop dx
530             jc movec2
531             ret
532 movec1:     pop dx
533             lea dx,errmsg3          ;'Read Error' message
534             ret
535 movec2:     lea dx,errmsg4          ;'Write Error' message
536             ret
537 move_cluster  endp
538 ;
539 ;---------------------------------------------------------------------------
540 ; SEARCH_DIR routine searches each entry in the disk directory image for a
541 ; starting cluster entry that matches the one input in DX.
542 ; Entry : DX - cluster number (2-XXX)
543 ; Exit  : BX - address of starting cluster word
544 ;         CF - clear : match found
545 ;              set   : not found
546 ;---------------------------------------------------------------------------
547 search_DIR    proc near
548             mov bx,DIR_data         ;get starting address of directory image
549             add bx,90               ;starting cluster offset - 3rd directory entry
550             mov cx,DIR_entries      ;get maximum number of search loops
551             sub cx,2                ;skip first two entries
552 sdir1:      cmp [bx],dx             ;do the words match for this entry?
553             je sdir2                ;yes, then clear CF and exit
554             add bx,32               ;no, then adjust BX for next entry
555             loop sdir1              ;try next entry
556             stc                     ;no match found
557             ret
558 sdir2:      clc                     ;match found
559             ret
560 search_DIR    endp
561 ;
562 endprog     label byte              ;start of buffer area
563 ;
564 code        ends
565             end begin
```

■

เป็น 2 ชุด ชุดละ 2 เซกเตอร์) และข้อมูลทั่วไปของผู้ใช้ 708 เซกเตอร์

การจองเนื้อที่บนดิสก์นั้นดอสกำหนดให้จองได้ในหน่วยของคลัสเตอร์ โดยที่หนึ่งคลัสเตอร์จะประกอบด้วยหนึ่งหรือสองเซกเตอร์ก็ขึ้นกับฟอร์แมตของดิสก์ และดิสก์ซึ่งมีความจุ 360 กิโลไบต์ จะมี 354 คลัสเตอร์ (หมายเลข 2-355) สำหรับเก็บข้อมูล เมื่อดอสจะทำการเก็บไฟล์ใหม่ลงบนดิสก์ ดอสจะต้องตรวจหาคลัสเตอร์ว่างเพื่อจะเก็บไฟล์ใหม่นั้น และหากมีดอสจะเขียนข้อมูลบนดิสก์ทีละเซกเตอร์ นอกจากนั้นดอสยังต้องเก็บไดเรคทอรีสำหรับไฟล์นั้นด้วยโดยแต่ละไดเรคทอรีจะประกอบด้วย ชื่อไฟล์, แอตทริบิวของไฟล์, ขนาดไฟล์, วันที่และเวลาที่สร้างไฟล์ การเขียนข้อมูลของไฟล์นั้นดอสจะไม่เขียนบนเซกเตอร์ที่ต่อเนื่องกันจนจบไฟล์ แต่อาจแบ่งไฟล์ออกเป็นส่วน ๆ แล้วเขียนบนเซกเตอร์ซึ่งยังว่างอยู่ทั้งนี้ก็เพื่อที่จะได้ใช้พื้นที่บนดิสก์ได้อย่างมีประสิทธิภาพ

ในการอ่านไฟล์จากดิสก์นั้นดอสจำเป็นต้องรู้ว่าไฟล์นั้นเก็บอยู่ที่ใดบนดิสก์บ้าง ซึ่งดอสก็ทำได้โดยอ่านตำแหน่งที่เก็บของข้อมูลจาก FAT ที่ดอสเก็บไว้ขณะที่สร้างไฟล์ครั้งแรก FAT จะเป็นตารางที่เก็บสถานะของแต่ละคลัสเตอร์ไว้ จากตารางที่ 1 จะแสดงความหมายของค่าที่เก็บใน FAT เช่น ถ้าค่าที่เก็บใน FAT เป็น 000 แสดงว่าคลัสเตอร์นั้นยังว่างอยู่ สามารถนำมาใช้ชี้ตำแหน่งของข้อมูลที่จะเก็บบนดิสก์เป็นต้น

สมมติว่าดอสต้องการเก็บข้อมูลโดยใช้เนื้อที่ 3 คลัสเตอร์ และดอสพบว่าคลัสเตอร์หมายเลข 5, 10, 11 เป็นคลัสเตอร์ว่าง ขณะที่เขียนไฟล์ดอสเก็บหมายเลขคลัสเตอร์ 10 ไว้ในตาราง FAT ตำแหน่งที่ 5, เก็บหมายเลขคลัสเตอร์ 11 ไว้ในตาราง FAT ตำแหน่งที่ 10 และเก็บค่า FFFh ไว้ในตาราง FAT ตำแหน่งที่ 11 จะเห็นว่าจุดเริ่มต้นของไฟล์นี้อยู่ที่คลัสเตอร์หมายเลข 5 ซึ่งดอสจะต้องเก็บหมายเลขคลัสเตอร์เริ่มต้นนี้ไว้ในไฟล์ไดเรคทอรีที่ออฟเซต 26 ดังนั้นดอสก็จะสามารถเขียนและอ่านข้อมูลได้โดยใช้ตาราง FAT และไฟล์ไดเรคทอรี

การจัดเนื้อที่บนดิสก์ใหม่

เนื่องจากดอสมีข้อกำหนดว่าในดิสก์ที่มีซิสเต็มนั้นจะต้องมี IBMBIO.COM และ IBMDOS.COM ที่ซ่อนไว้ ซึ่งไฟล์ทั้งสองนี้ต้องเก็บไว้ในดิสก์ไดเรคทอรีสองตำแหน่งแรก นอกจากนี้ IBMBIO.COM ยังต้องเก็บในเซกเตอร์ที่อยู่ต่อเนื่องกันโดยเริ่มต้นจากคลัสเตอร์แรกของแผ่นด้วย

PSYS จะต้องย้ายข้อมูลในดิสก์ไดเรคทอรีสองตำแหน่งแรกออกก่อนให้เกิดที่ว่างเพื่อให้ SYS ทำงานได้โดยจะเรียก FIND_ENTRY มาค้นหาที่ว่าง จากนั้นจะเรียก MOVE_ENTRY มาย้ายข้อมูลไปไว้ที่ว่างนั้น เนื่องจากข้อจำกัดของ IBMBIO.COM ที่ต้องเริ่มต้นเก็บที่คลัสเตอร์แรกและต้องต่อเนื่องกันทั้งไฟล์ด้วย เราจึงจำเป็นต้องย้ายข้อมูลทั้งหมดที่อยู่ในบริเวณนี้ออกให้หมด โดยเราจะกันเนื้อที่ไว้ 40 คลัสเตอร์ ซึ่งเผื่อไว้สำหรับ DOS รุ่นใหม่ๆ ที่อาจมีขนาดไฟล์ใหญ่

การย้ายข้อมูลจะเริ่มต้นอ่านค่าจากตาราง FAT ของคลัสเตอร์หมายเลข 2 มาตรวจดูว่า ถ้ามีค่าอยู่ระหว่าง FF0h-FF7h แสดงว่าเป็นคลัสเตอร์เสีย หรือ ถ้ามีค่าเป็นศูนย์แสดงว่าเป็นคลัสเตอร์ว่าง แต่ถ้าเป็นค่าอื่นก็แสดงว่าคลัสเตอร์นี้มีข้อมูลเก็บอยู่ ดังนั้นเราจะต้องย้ายข้อมูลนี้ออก โดยจะเรียก FIND_CLUSTER ให้ค้นหาคลัสเตอร์ว่างที่ไม่อยู่ใน 40 คลัสเตอร์ที่เรากันไว้ จากนั้น MOVE_CLUSTER จะย้ายข้อมูลทั้งหมดไปไว้ที่คลัสเตอร์ว่างนั้น แล้วจะเอาค่าจากตาราง FAT ของคลัสเตอร์หมายเลข 2 มาเก็บไว้ใน FAT ในตำแหน่งของคลัสเตอร์ว่างนั้น จากนั้นก็จะเซตให้ FAT ของคลัสเตอร์หมายเลข 2 เป็นศูนย์

สุดท้ายเราจะต้องค้นหาว่ามีค่าใน FAT หรือ ในไฟล์ไดเรคทอรี (ออฟเซตที่ 26) ใดบ้างที่อ้างถึงคลัสเตอร์หมายเลข 2 ซึ่งเราจะต้องแก้ให้ค่าของ FAT นั้นชี้ไปที่คลัสเตอร์ว่างที่ FIND_CLUSTER ค้นพบได้แทน หากไม่มีข้อผิดพลาดใดเกิดขึ้นโปรแกรมก็จะกลับไปอ่านค่าจาก FAT ของคลัสเตอร์หมายเลขต่อไปมาจนครบ 40 คลัสเตอร์

ก่อนจบ

PSYS จะช่วยจัดเนื้อที่บนดิสก์ใหม่สำหรับให้ SYS ย้ายซิสเต็มไฟล์มายังดิสก์นั้น สำหรับข้อจำกัดในการใช้ PSYS ก็คือ PSYS จะทำงานกับฟลอปปี้ดิสก์เท่านั้น, ไม่ควรใช้ PSYS กับดิสก์ที่ป้องกันการก๊อบปี้ และดิสก์ที่มีไดเรคทอรีย่อย

ตารางที่ 1 แสดงค่าและความหมายที่ดอสเก็บใน FAT

ค่าใน FAT	ความหมายของดอส
000	คลัสเตอร์ว่างสำหรับเก็บไฟล์ใหม่
FF0-FF6	คลัสเตอร์ที่ดอสจองไว้
FF7	คลัสเตอร์เสีย
FFF	คลัสเตอร์สุดท้ายของไฟล์
อื่น ๆ	คลัสเตอร์ถูกใช้เก็บข้อมูลอยู่